Multimedia-Programmierung
Übung 9

Ludwig-Maximilians-Universität München

Sommersemester 2014

Today

- State Machines in 
- Physics
- The final Project

AI in Games

- Intelligent behavior (e.g. decision making) makes characters in games more realistic
- AI in games: decide on current knowledge and state, which steps to take next
- Examples: Enemy only attacks player in certain range, Sims decide on their next activity based on current mood

State Machines

consist of:

- states
- start state

- state actions
- entry and exit actions

- transitions
- transition conditions

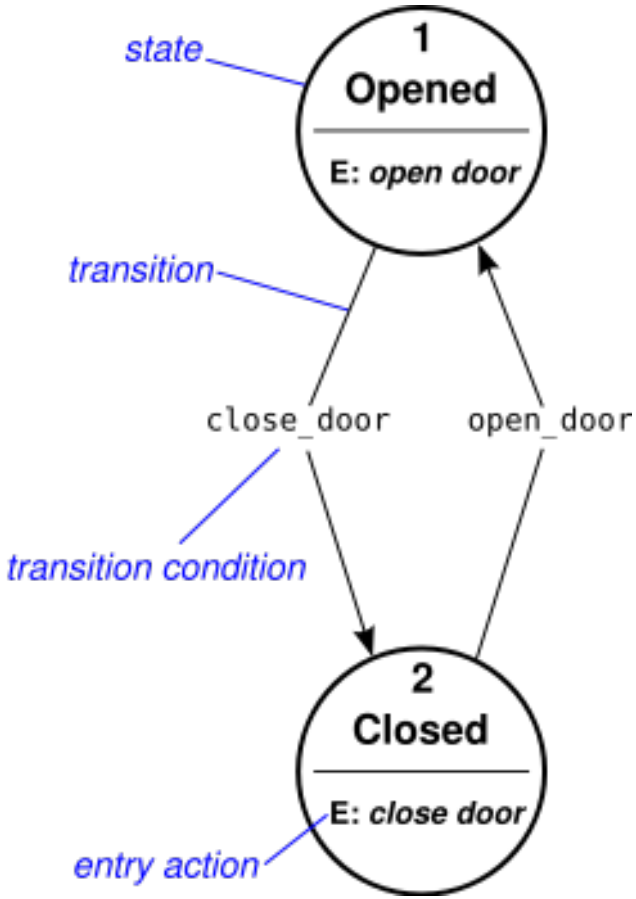
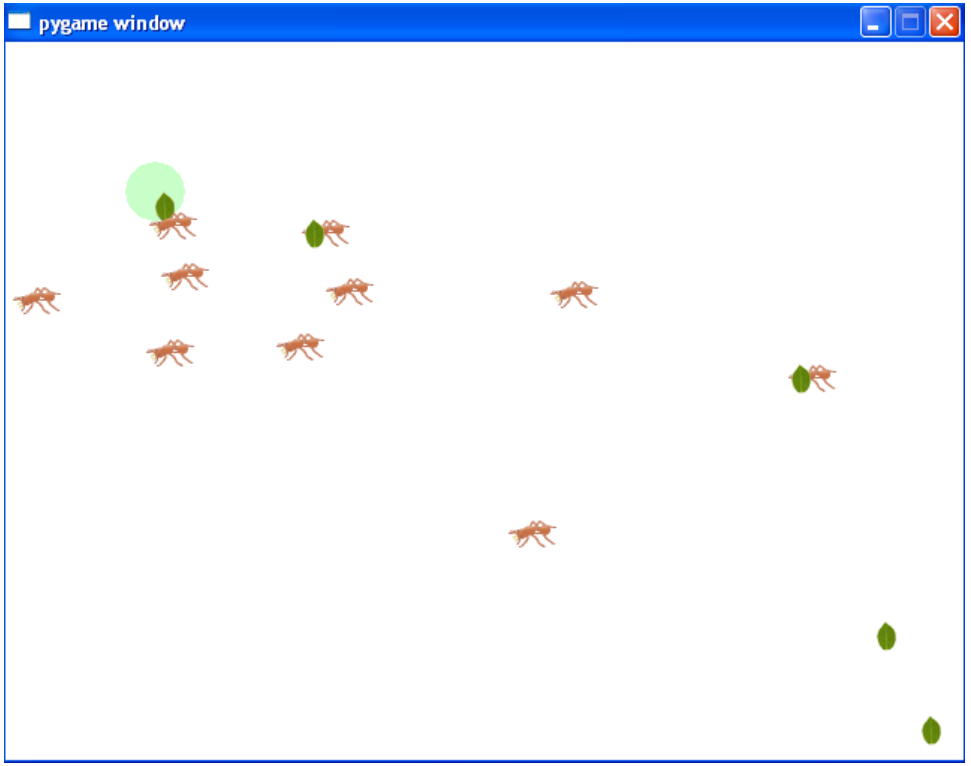


image source: Wikipedia

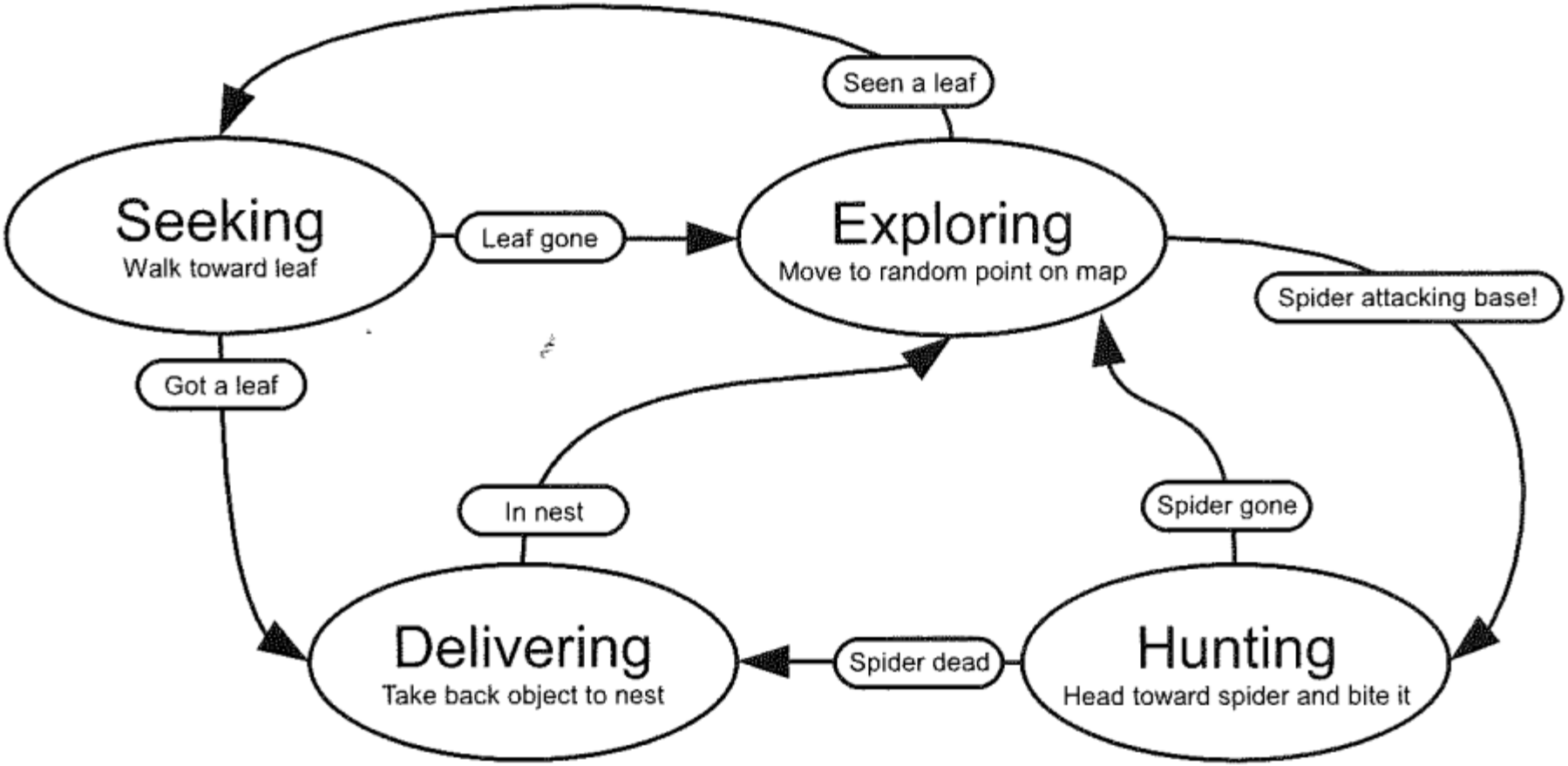
Example: Ant Nest

- Ants search for food and deliver it to their nest



Example from Book „Beginning Game Development with Python and Pygame – From Novice to Professional“ by Will McGugan

Example: Ant Nest



State Machine

```
class StateMachine(object):
    def __init__ (self):
        self.states = {}
        self.active_state = None

    def add_state(self, state):
        self.states[state.name] = state

    def think(self):
        if self.active_state is None:
            return

        self.active_state.do_actions()

        new_state_name = self.active_state.check_conditions()
        if new_state_name is not None:
            self.set_state(new_state_name)

    def set_state(self, new_state_name):
        if self.active_state is not None:
            self.active_state.exit_actions()

        self.active_state = self.states[new_state_name]
        self.active_state.entry_actions()
```

List of states

is called in every update(..)

1. Do actions for current state
2. Check if state changed
3. Eventually change state

4. Do exit actions for old state
5. Do entry actions for new state

State

```
class State(object):
    def __init__ (self, name):
        self.name = name

    def do_actions(self):
        pass

    def check_conditions(self):
        pass

    def entry_actions(self):
        pass

    def exit_actions(self):
        pass
```

Actions in this state (e.g. update animation, walk somewhere etc.)

Check conditions for this state and eventually change to another state

If changed to this state, do specific actions

If current state gets inactive, do some exit actions

State Example

```
class State(object):
    def __init__(self, name):
        self.name = name

    def do_actions(self):
        pass

    def check_conditions(self):
        pass

    def entry_actions(self):
        pass

    def exit_actions(self):
        pass
```

```
import random
class AntStateExploring(State):
    def __init__(self, ant):
        State.__init__(self, "exploring")
        self.ant = ant

    def do_actions(self):
        #change direction in approx. every 20th call
        if random.randint(1, 20) == 1:
            self.random_destination()

    def check_conditions(self):
        leaf = self.ant.world.get_close_entity("leaf", self.ant.location)
        if leaf is not None:
            self.ant.leaf_id = leaf.id
            return "seeking"
        return None

    def entry_actions(self):
        self.ant.speed = 120. + random.randint(-30,30)
        self.random_destination()

    def random_destination(self):
        ...
```

Other useful classes for game development

BaseClass for **Game Entities**:

- Moving the game entity
- Rendering the game entity
- Updating current state
- Etc.

```
class GameEntity(object):
    def __init__(self, world, name, image, initial_position):
        self.world = world
        self.name = name
        self.image = image
        self.location = initial_position
        self.destination = (0,0)
        self.speed = 0.
        self.brain = StateMachine()
        self.id = 0

    def render(self, surface):
        x,y = self.location
        w, h = self.image.get_size()
        surface.blit(self.image, (x-w/2, y-h/2))

    def process(self, time_passed):
        self.brain.think()
#calculate new position and move game entity
        …
```

Other useful classes for game development

World:

- Stores all game entities (e.g. in a dictionary) and assigns IDs to new entities
- Starts update and rendering process of entities
- Can provide queries for entities (e.g. entities in range etc.)

```
class World(object):
    def __init__(self):
        self.entities = {}
        self.entity_id = 0
self.background = …

    def add_entity(self, entity):
        self.entities[self.entity_id] = entity
        entity.id = self.entity_id
        self.entity_id += 1

    def remove_entity(self, entity):
        del self.entities[entity.id]

    def get(self, entity_id):
        …
```

Other useful classes for game development

```
…
    def process(self, time_passed):
        time_passed_seconds = time_passed/1000.0
        for entity in self.entities.values():
            entity.process(time_passed_seconds)

    def render(self, surface):
        surface.blit(self.background, (0,0))
        for entity in self.entities.values():
            entity.render(surface)

    def get_close_entity(self, name, location, range=100):
        …
```

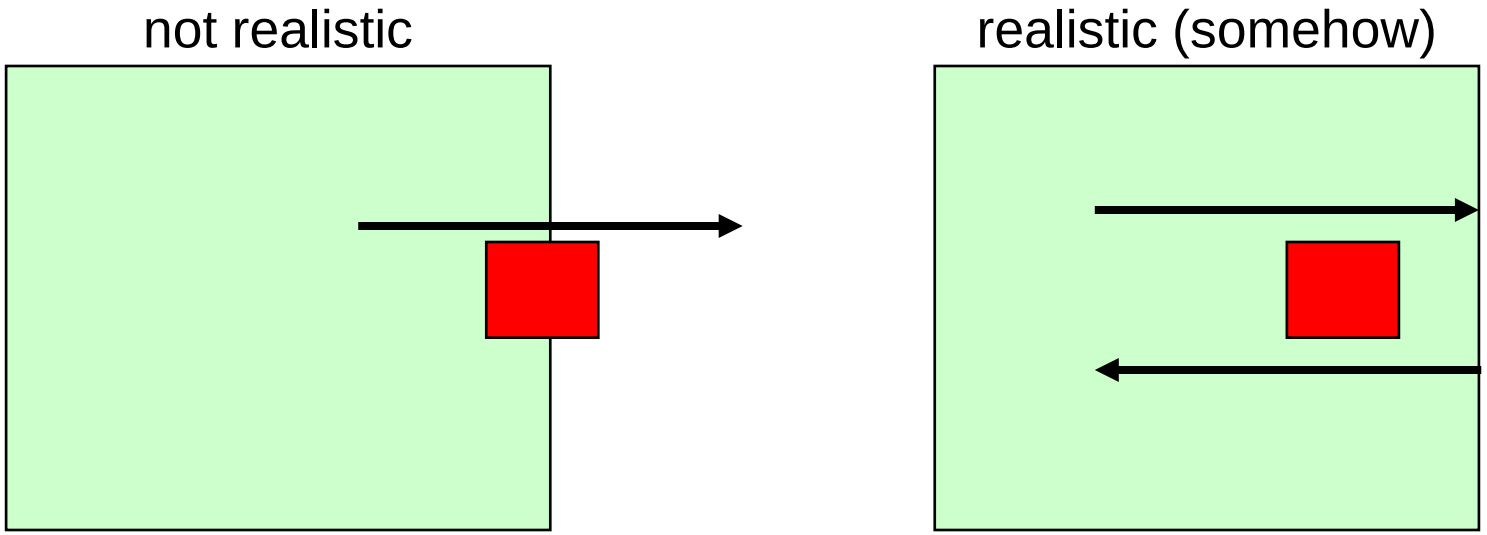
Physics

How logical behaviour improves usability

Users have specific expectations

For example, if something hits a wall it should bounce or create some damage

Adding physics to applications helps to improve usability



Physics

Examples I - Bumptop

A physically enhanced Windows desktop

©bumptop.com

Physics

Examples II - Physics and Microsoft Surface

Allows physically correct interaction with a tabletop device

Wilson, A. D., Izadi, S., Hilliges, O., Garcia-Mendoza, A., and Kirk, D. 2008. Bringing physics to the surface. In Proceedings of the 21st Annual ACM Symposium on User interface Software and Technology (Monterey, CA, USA, October 19 - 22, 2008). UIST '08. ACM, New York, NY, 67-76.

Programming Physics

Frameworks, APIs, development tools etc. often offer physics engines (e.g. 3D game engines, Interpolators in Flash or Box2D for JavaScript (..and python))

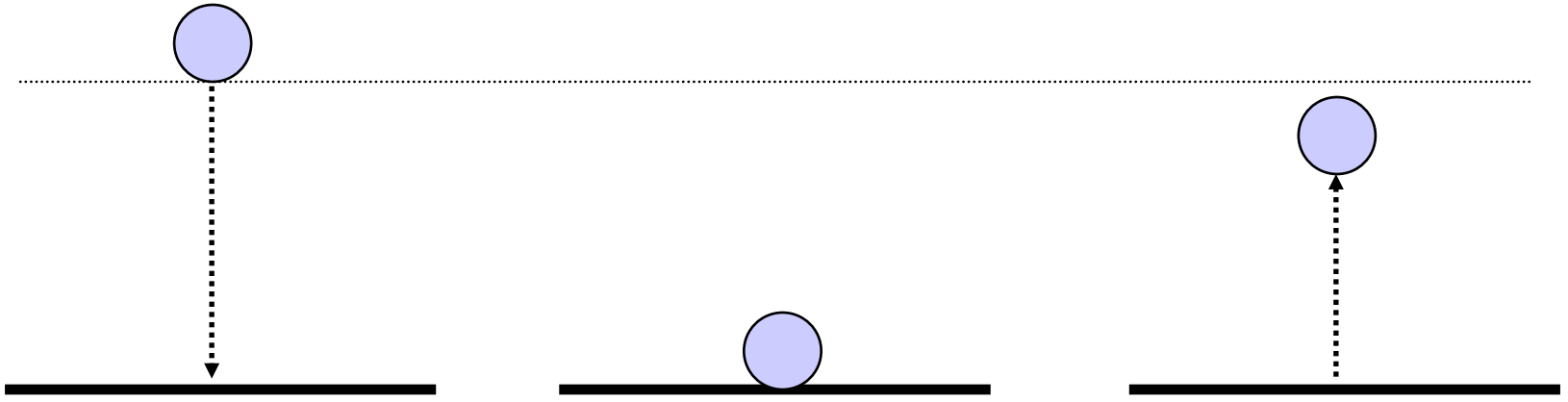
In Python, (usually) **WE** do the physics!!

Tutorial:
http://physics.gac.edu/~miller/jterm_2013/physics_engine_tutorial.html

Bouncing Ball Example 1

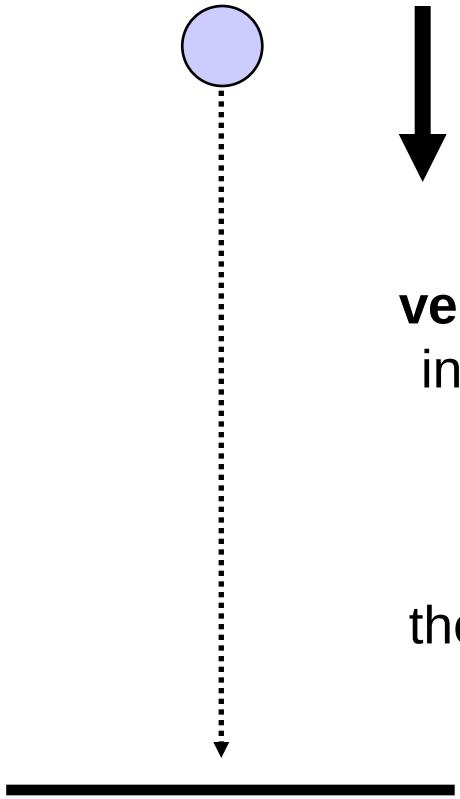
Let’s make a ball bounce in a realistic way

1. We need a concept:



Bouncing Ball Example 2

2. What makes the ball fall and bounce?



gravity makes the ball fall

velocity depends on gravity and increases/decreases over time

the material of the ball influences how far it will **bounce** back

Bouncing Ball Example 3

```
class Ball(pygame.sprite.Sprite):
    def __init__(self, color, initial_position):
        pygame.sprite.Sprite.__init__(self)
        size = 20
        self.gravity = 900
        self.velocity = 0
        self.bounce = 0.9

        self.image = pygame.Surface((size,size),pygame.SRCALPHA,32)
        pygame.draw.circle(self.image,color,(size/2,size/2),size/2)
        self.rect = self.image.get_rect()
        self.rect.center = initial_position

    def update(self, time_passed, size):

        self.velocity += (self.gravity * time_passed)
        self.rect.bottom += int(self.velocity * time_passed)

        if self.rect.bottom >= size[1]:
            self.rect.bottom = size[1]
            self.velocity = -self.velocity * self.bounce
```

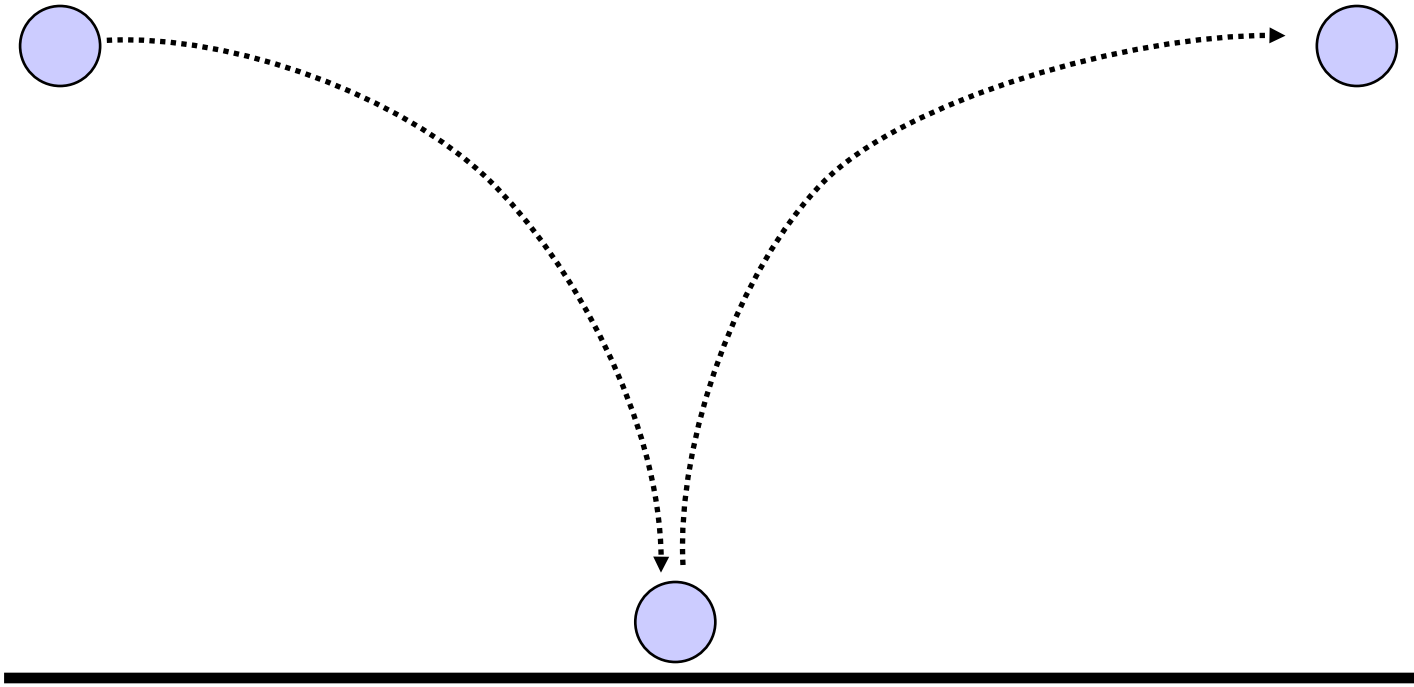
gravity per second, current velocity and bounce factor of the material

velocity is increased/decreased by the gravity

if the ball hits the ground, reduce velocity based on the bounce factor

Bouncing Ball Example 4

Making the ball bounce and move vertically



Bouncing Ball Example 5

```
class Ball(pygame.sprite.Sprite):
    def __init__(self, color, initial_position):
        pygame.sprite.Sprite.__init__(self)
        size = 20
        self.gravity = 900
        self.vx = 0
        self.vy = 0
        self.bounce = 0.9
        …
    def update(self, time_passed, size):
        self.velocity += (self.gravity * time_passed)
        ydistance = int(self.vy * time_passed)
        self.rect.bottom += ydistance
        if ydistance == 0 and self.rect.bottom == size[1]: self.vx = 0
        self.rect.left += int(self.vx * time_passed)
        if self.rect.right >= size[0]:
            self.rect.right = size[0]
            self.vx = -self.vx
        if self.rect.left <= 0:
            self.rect.left = 0
            self.vx = -self.vx
        if self.rect.bottom >= size[1]:
            self.rect.bottom = size[1]
            self.vy = -self.vy* self.bounce
```

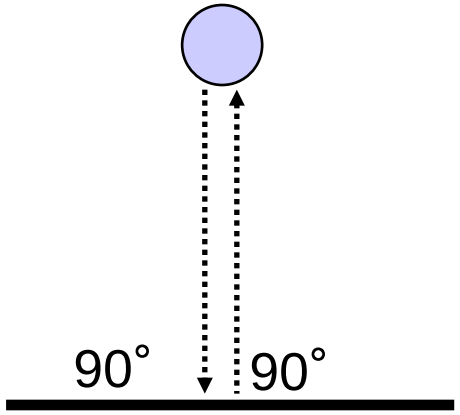
x and y velocity

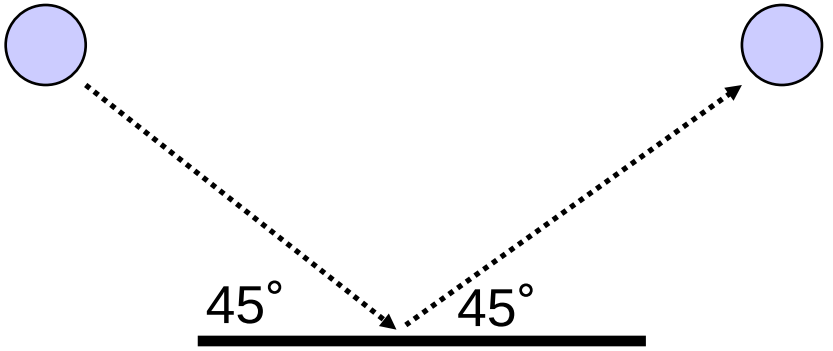
clumsy way to make the ball stop

if the ball hits the sidewalls, make it change the direction

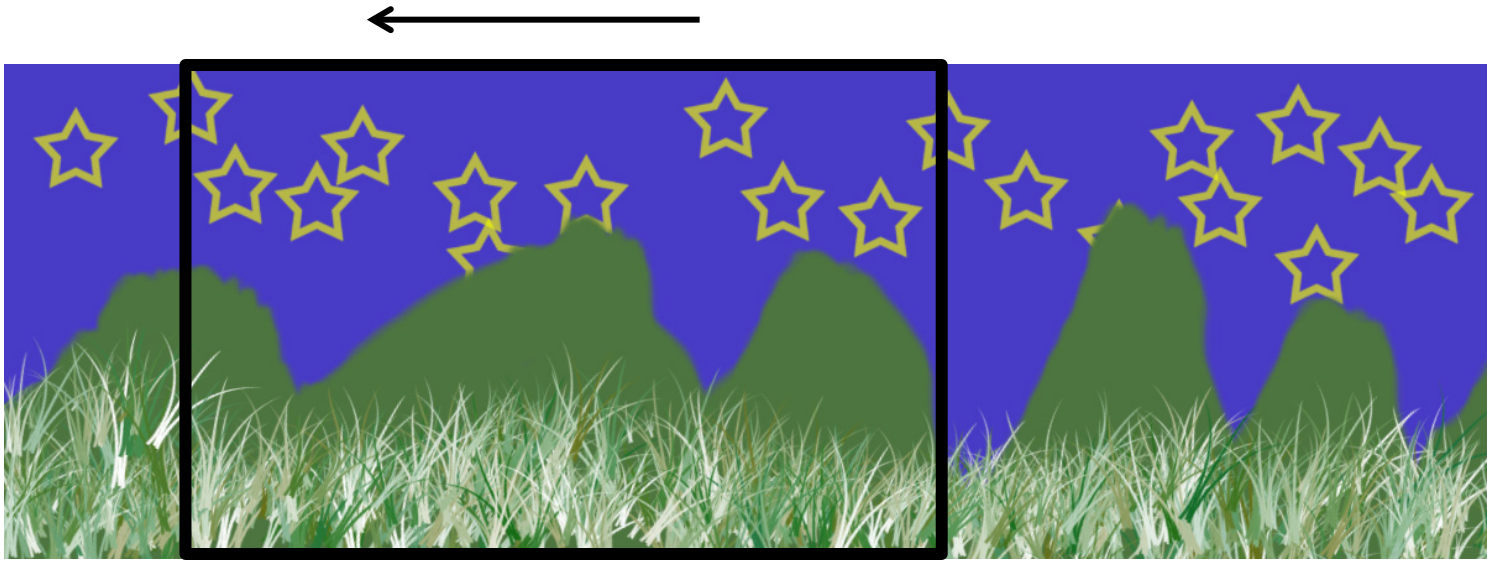
Arrival Angle = Angle of Reflection

What if the Ball doesn't drop perfectly vertically?





Motion: Scrolling Background



When the player moves, the world moves in the opposite direction.

Scrolling Background

From: http://bytesforlunch.wordpress.com/

```
clock = pygame.time.Clock()
screen = pygame.display.set_mode((600,400),0,32)

b1 = "back.jpg“
back = pygame.image.load(b1).convert()
back2 = pygame.image.load(b1).convert()
x = 0
screenWidth = 1200

while True:
        for event in pygame.event.get():
                if event.type == QUIT:
                        pygame.quit()
                        sys.exit()

        screen.blit(back, (x,0))
        screen.blit(back2,(x+screenWidth,0))

        x = x - 1
        if x == -screenWidth:
                x = 0

        msElapsed = clock.tick(100)
        pygame.display.update()
```

The viewport

Copying graphics to the screen

Display the graphics

Parallax-Scrolling

Parallax effect

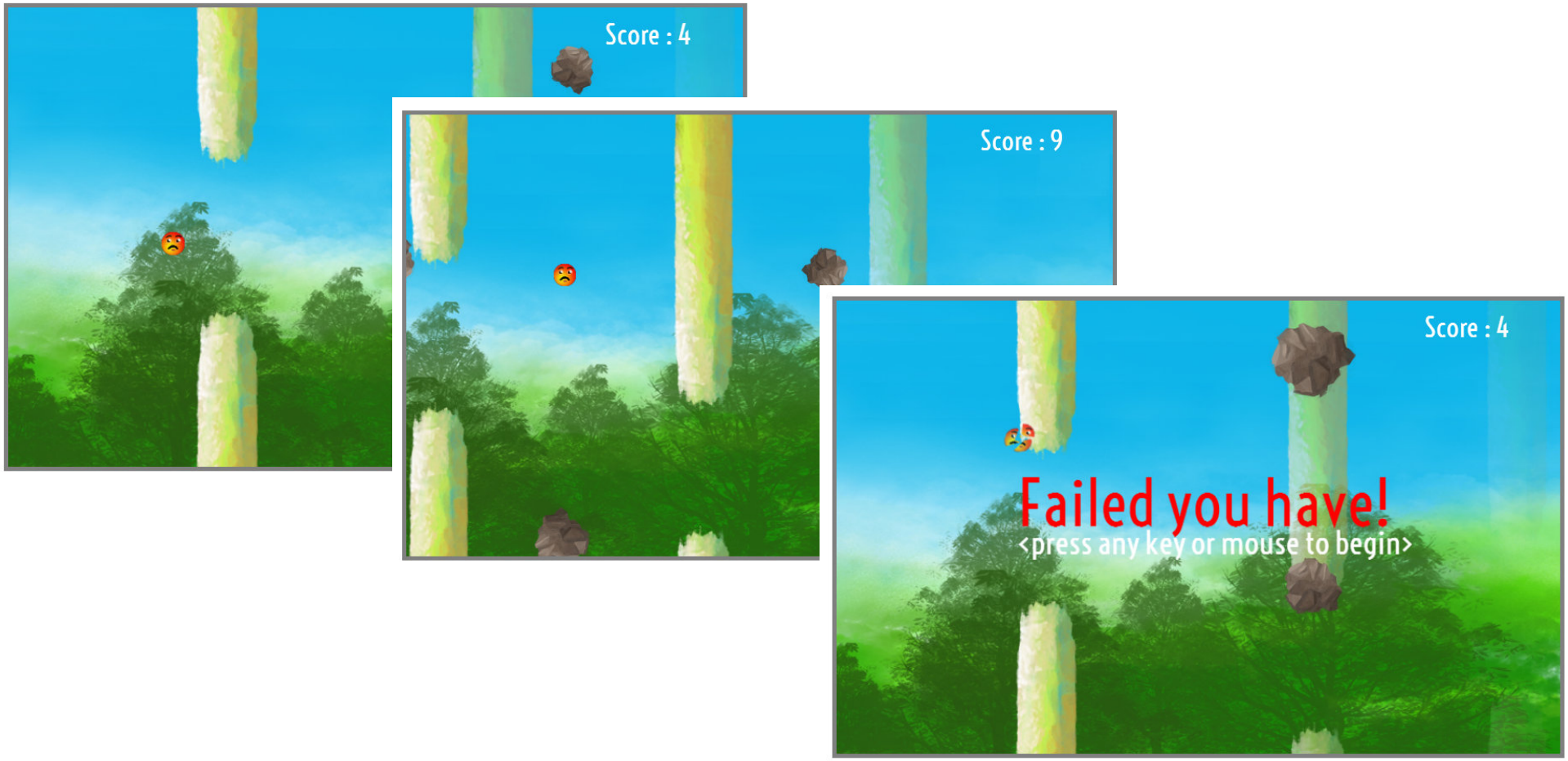
distant objects appear to be moving slower than closer objects

➡ Layers: changing layers at different speed
Sprites: individual movable objects (pseudo layers)

© https://github.com/joshbyrom/PyScrolling.git; joshbyrom

Final Project: “Flappy Bird”

Example by Lukas Mecke



Final Project: “Flappy Bird”

Characteristics

Origin:

- Originally developed by Dong Nguyen (2013) in three days
- Side-scrolling 2D Game

Gameplay:

- Fly a continuously moving character trough obstacles
- Collisions end the game
- Character flaps each time a button is pressed
- Character falls if no button is pressed for some time

Final Project: “Flappy Bird”

Requirements

- Design and Implement your own clone of “Flappy Bird”
- Submissions will be reviewed and can **gain up to 10% bonus for the final exam**
- Games can be developed in Python, JavaFX and JavaScript
- Project Phase: 16.06. – 14.07.

! All submissions will be checked for plagiarism!